AN ABSTRACT OF THE THESIS OF

Hu Zhou for the degree of Master of Science in Computer Science presented on June 22, 2000.

Title: Content-based Multicast in Ad Hoc Networks

Redacted for privacy

Abstract approved: _____

Prof. Bella Bose

An important objective of tactical ad hoc networks is to deliver threat information from sensors to shooters efficiently and quickly. The information sent to a particular shooter should contain warnings about threats that are within some distance and/or within some time of the shooter's current location. In this thesis we develop a novel multicast model that distributes this form of threat information in a message efficient manner. In addition, information about allied force can also be distributed in a similar way. We present results from extensive simulations that demonstrate the efficiency of our protocol and discuss the scalability of this model to larger networks.

Content-based Multicast in Ad Hoc Networks

by

Hu Zhou

A Thesis

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed June 22, 2000
Commencement June 2001

Master of Science thesis of Hu Zhou presented on June 22, 2000

APPROVED:

Redacted for privacy

Major Professor, representing Computer Science

Redacted for privacy

Chair of the Department of Computer Science

Redacted for privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for privacy

Hu Zhou, Author

ACKNOWLEDGMENTS

My earnest thanks go to Dr. Suresh Singh, who has provided the original idea of this research and has given me constant guidance throughout the process. My thanks are also extended to Pete Eberlein who wrote the visual display part of the simulation and provided some very useful suggestions. I'd also like to thank Professor Bella Bose.

TABLE OF CONTENTS

		Page
Chapter 1:	Introduction	1
1.1	Ad Hoc Network	1
1.2	Multicast in Ad Hoc Network	3
1.3	Time-Space Multicast Support	4
Chapter 2:	Related Work	6
2.1	Ad hoc Multicast Routing(AMRoute)	6
2.2	On-Demand Multicast Routing Protocol (ODMRP)	8
2.3	Ad hoc Multicast Routing Protocol Utilizing Increasing ID Numbers (AMRIS)	10
2.4	Core-Assisted Mesh Protocol (CAMP)	11
Chapter 3:	Overview of the Protocol	13
3.1	Assumptions	13
3.2	Scenario	13
3.3	Difference Between the Time-Space and Other Multicast Models	15
3.4	Push/Pull - the Basic Strategy	16
Chapter 4:	Details of the Protocol	17
4.1	Partitioning the Field	17
4.2	Election of Agent	17
4.3	Agent Maintenance	18
4.4	Routing Algorithm	19
4.5	Generating Threat Information	21
4.6	Push Protocol	21

TABLE OF CONTENTS (Continued)

Page

4.7 Pull Protocol 23

4.8 Sticky Pull 27

4.9 Reply/Updates - Completing the Dissemination 28

4.10 Ally Information Dissemination 29
4.10.1 Pushes 29
4.10.2 Pull, Reply and Update 30

Chapter 5: Performance Evaluation 31

5.1 The Simulation Program 31

5.2 Performance Metrics 34

5.3 Simulation Parameters 35

5.4 Discussion of Results 36

Bibliography 44

LIST OF FIGURES

Figure		Page
1.1	A simple ad hoc network. Two nodes communicate with the help of another	2
1.2	A multicast group consists of nodes A, B, C and D	5
4.1	An example of partioning the field - 8 blocks	18
4.2	An example of MFR	20
4.3	An example of push	23
4.4	pull area calculation	26
4.5	Accumulative Pull	26
5.1	Node Classes Hierarchy	32
5.2	Message overhead – Tanks only.	37
5.3	Message overhead – Shooters only.	37
5.4	Threat Information Success Rate – Tanks only.	38
5.5	Threat Information Success Rate – Shooters only.	39
5.6	Ally Force Information Success Rate – Tanks only.	40
5.7	Ally Force Information Success Rate – Shooters only.	40
5.8	Threat Information Success Rate for Different Block Size.	41
5.9	Message Overhead for Different Block Size.	42
5.10	Ally Force Information Success Rate for Different Block Size.	42

CONTENT-BASED MULTICAST IN AD HOC NETWORKS

Chapter 1

INTRODUCTION

1.1 Ad Hoc Network

Mobile hosts and wireless networking hardware are becoming widely available, and there will be an increasingly extensive use of such devices, in many applications. Extensive work has been done recently in integrating these elements into traditional networks such as the Internet. Often, however, mobile users want to communicate in situations in which no fixed wired infrastructure such as this is available, either because it may not be economically practical or physically possible to provide the necessary infrastructure or because the expediency of the situation does not permit its installation. In such case, the hosts form an ad hoc network.

An ad hoc network is a collection of wireless mobile hosts forming a temporary network without the aid of any established infrastructure or centralized administration. Typical situations are disaster relief, and battlefield.

Ad-hoc networks differ significantly from existing networks. First of all, the topology of interconnections may be quite dynamic. Secondly, most users will not wish to perform any administrative actions to set up such a network. In order to provide service in the most general situation, we do not assume that

every computer is within communication range of every other computer. This lack of complete connectivity would certainly be a reasonable characteristic of, say, a population of mobile computers in a large room that relied on infrared transceivers to effect their data communications.

From a graph theoretic point of view, an ad-hoc network is a graph, $G(N, E(t))$, which is formed by denoting each mobile host by a node and drawing an edge between two nodes if they are in direct communication range of each other. The set of edges, $E(t)$, so formed, is a function of time, and it keeps changing as nodes in the ad-hoc network move around. The topology defined by such a network can be very arbitrary since there are no constraints on where mobiles could be located with respect to each other.

In such an environment, it may be necessary for one mobile host to enlist the aid of other hosts in forwarding a packet to its destination, due to the limited range of each mobile host's wireless transmissions. So each host functions as a router too. In figure 1.1, a simple example of forwarding is illustrated. Node A needs to communicate with node C, but they are outside of each other's transmission range. In this case, they can ask node B to be the intermediary node to forward their communication, which can also talk to both A and C.

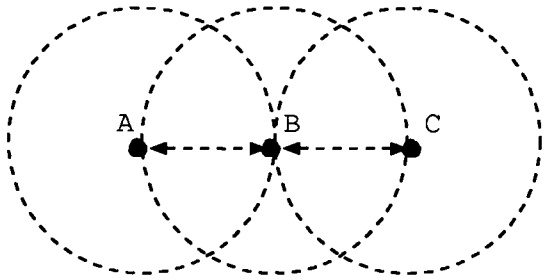

FIGURE 1.1: A simple ad hoc network. Two nodes communicate with the help of another

1.2 Multicast in Ad Hoc Network

Wireless information services are characterized by their geographic scope. In an ad hoc network, usually the information content and the methods of the information delivery depend on the location of the user. Wide-area services such as stock market information, will be offered on a national scale. Macroservices, such as weather, would be provided on a regional basis, with regions extending to tens or hundreds of miles. The geographic scope of microservices such as traffic conditions will extend to the size of that of one cell in a cellular network. Finally a picoservices is for an area in size of hundred meters in diameter. For example, parking availability could be provided within such a scope.

In a typical ad hoc environment, network hosts work in groups to carry out a given task. Hence multicast plays an important role in ad hoc networks. For example, disaster relievers in a disaster site, and soldiers in a battlefield, all need to keep contact with each other in a whole group, and share critical information among them, to accomplish their goal. In a battlefield, information about an enemy tank group should be sent to the friendly tanks and soldiers nearby, but not to those far away. And also this information is not interesting to bombers passing by above because tanks impose no threat to them. So hosts in the ad hoc network are usually divided into different multicast groups, based on their location and features. Hosts may be grouped according to their location, mobility, and function. Hosts in a group can be aggregated geographically, or they can be spread all over the battlefield. A multicast protocol is responsible for organizing and maintaining the multicast groups, and disseminating information within each multicast group.

Much difference exists between multicast in ad hoc network and multicast

in wired network. In wired networks, once the hosts are stationary and once the group is defined, it is easy to maintain. For example in tree-based multicast, once the multicast tree is built for a group, it remains relatively stable. Hosts' joining and leaving a group does not bring much cost. However in an ad hoc network, the hosts are mobile, usually in a rather random pattern. A multicast tree that consists of such hosts would be more difficult to maintain. And as we will see later, sometimes the membership of a host to a group depends on its own mobility pattern.

1.3 Time-Space Multicast Support

An important application of tactical Ad Hoc network is to deliver information about enemies and friendly nodes in an efficient and prompt way. The Time-Space Multicast Protocol aims to provide a message delivery mechanism in such a scenario. It makes the proper information flow to units based on their specific requirements. Information should be disseminated only to nodes that are in need of it, otherwise part of the bandwidth and power in sending the information will be wasted. The defining feature of the time-space multicast protocol is that the message is disseminated to hosts based on their requirements.

Now we look at information about what threats and other friendly nodes are useful to an individual host. As a host moves in the battlefield, it may enter one or more threats' strike ranges. It is dangerous for it to be ignorant of that fact. It is highly helpful to get information about the enemies that may pose a threat. Typically a host is concerned with those enemies it will encounter during a certain period of time or within a certain distance. Thus we define two parameters for a host: the time parameter and the space parameter, to specify

the host's need for information. Multicast: for each threat, the friendly hosts that are going to be threatened compose a multicast group. Proper information should be disseminated to the members in the group. Under such a definition, the group membership is dependent on the mobility of both the friendly node itself and the threat. Figure 1.2 shows an example of a multicast group of friendly tanks that are threatened by one enemy tank. In this example, host A, B, C and D are threatened by the threat. The sensor that detects the threat should deliver the information to the four nodes. Here note that simply flood the warning to the area which the four nodes are in is not an efficient way, because many other nodes in that area may be heading in other direction and thus do not need such warning message.

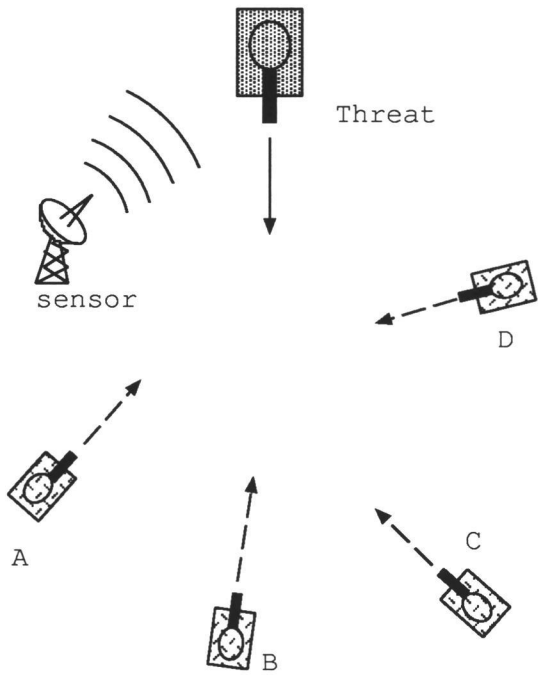

FIGURE 1.2: A multicast group consists of nodes A, B, C and D

Chapter 2

RELATED WORK

Multicast protocols used in static networks, e.g., Distance Vector Multicast Routing Protocol(DVMRP)[2], Multicast Open Shortest Path First(MOSPF) [9], Core Based Trees(CBT) [14], and Protocol Independent Multicast(PIM) [10] do not perform well in ad hoc networks because multicast tree structures are fragile and must be readjusted as connectivity changes. Furthermore, multicast trees usually require a global routing substructure such as link state or distance vector. The frequent exchange of routing vectors or link state tables, triggered by continuous topology changes, yields excessive channel and processing overhead and frequent loss of datagrams.

Various multicast protocols have been proposed to perform multicasting in ad hoc networks. Here we select five of them and describe how they work.

2.1 Ad hoc Multicast Routing(AMRoute)

The first one is Ad Hoc Multicast Routing, or AMRoute [3]. This is a tree-based protocol. Using unicast channels in the network, a shared tree is created for multicast group members to share information. The tree is bi-directional. One and only one tree is needed for one multicast group. Each group has at least one logical core that is responsible for member and tree maintenance. This logical core is significantly different from the core in CBT and the RP in PIM–SM

as it is not the central point on the data path (only for signaling), it is not a preset node (chosen from among currently known members) and it can change dynamically. The multicast tree includes only the group senders and receivers as its nodes. Each node is aware of its tree neighbors, and forwards data on the tree links. Multicast state is maintained by the group nodes only, and is not required by other network nodes.

The bi-directional multicast tree is obtained in two steps: the mesh creation and the tree creation. Initially, each group member declares itself as a core for its own group of size one. Each core periodically floods JOIN-REQS (using an expanding ring search) to discover other disjoint mesh segments for the group. When a member node receives a JOIN-REQ from a core of the same group but a different mesh segment, it replies with a JOIN-ACK and marks that node as a mesh neighbor. The node that receives a JOIN-ACK also marks the sender of the packet as its mesh neighbor. When the process is done, all nodes in the multicast group are connected into a mesh. Each mesh link is a unicast channel. Then each core periodically transmits TREE-CREATE packets to mesh neighbors in order to build a shared tree. When a member node receives a non-duplicate TREE-CREATE from one of its mesh links, it forwards the packet to all other mesh links. If a duplicate TREE-CREATE is received, a TREE-CREATE-NAK is sent back along the incoming link. The node receiving a TREE-CREATE-NAK marks the link as mesh link instead of tree link. The nodes wishing to leave the group send the JOIN-NAK to the neighbors and do not forward any data packets for the group.

The key characteristic of AMRoute is its usage of virtual mesh links to establish the multicast tree. The virtual mesh link is implemented based on a unicast. Therefore, as long as routes between tree members exist via mesh links,

the tree need not be readjusted when network topology changes. Non-members do not forward data packets and need not support any multicast protocol. Thus, only the member nodes that form the tree incur processing and storage overhead. AMRoute relies on an underlying unicast protocol to maintain connectivity among member nodes and any unicast protocol can be used. Like DVMRP [2] , CBT and PIM the protocol is independent from specific semantics of the underlying unicast routing protocol.

The major disadvantage of the protocol is that it suffers from temporary loops and creates non-optimal trees when mobility is present.

2.2 On-Demand Multicast Routing Protocol (ODMRP)

ODMRP [11], [12], [13], creates a mesh of nodes (the "forwarding group") which forward multicast packets via flooding (within the mesh), thus providing path redundancy. ODMRP is an on-demand protocol, thus it does not maintain route information permanently. It uses a soft state approach in group maintenance. Member nodes are refreshed as needed and do not send explicit leave messages. In ODMRP, group membership and multicast routes are established and updated by the source on demand. Similar to on demand unicast routing protocols, a request phase and a reply phase comprise the protocol. When multicast sources have data to send, but do not have routing or membership information, they flood a JOIN DATA packet. When a node receives a non-duplicate JOIN DATA, it stores the upstream node ID (i.e., backward learning) and rebroadcasts the packet. When the JOIN DATA packet reaches a multicast receiver, the receiver creates a JOIN TABLES and broadcast to the neighbors. When a node receives a JOIN TABLE, it checks if the next node ID of one of

the entries matches its own ID. If it does, the node realizes that it is on the path to the source and thus is part of the forwarding group. It then broadcasts its own JOIN TABLE built upon matched entries. The JOIN TABLE is thus propagated by each forwarding group member until it reaches the multicast source via the shortest path. This process constructs (or updates) the routes from sources to receivers and builds a mesh of nodes, the forwarding group. Multicast senders refresh the membership information and update the routes by sending JOIN DATA periodically. In networks where GPS (Global Positioning System) [4] is available, ODMRP can be made adaptive to node movements by utilizing mobility prediction [13]. By using location and mobility information supported by GPS, route expiration time can be estimated and receivers can select the path that will remain valid for the longest time. With the mobility prediction method, sources can reconstruct routes in anticipation of route breaks. This way, the protocol becomes more resilient to mobility. The price is, of course, the cost and additional weight of GPS. The details of mobility prediction and the procedure are described in [13]. The data transfer phase is identical for both versions. Nodes forward the data if they are forwarding nodes and the packet they receive is not a duplicate. Since all forwarding nodes relay data, redundant paths (when they exist) can help deliver data when the primary path becomes disconnected because of mobility. Another unique property of ODMRP is its unicast capability. Not only can ODMRP coexist with any unicast routing protocol, it can also operate very efficiently as unicast routing protocol. Thus, a network equipped with ODMRP does not require a separate unicast protocol.

2.3 Ad hoc Multicast Routing Protocol Utilizing Increasing ID Numbers (AMRIS)

AMRIS [1] establishes a shared tree for multicast data forwarding. Each node in the network is assigned a multicast session ID number. The ranking order of ID numbers is used to direct the flow of multicast data. Like ODMRP, AMRIS does not require a separate unicast routing protocol. Initially, a special node called Sid broadcasts a NEW-SESSION packet. The NEW-SESSION includes the Sid's msm-id (multicast session member id). Neighbor nodes, upon receiving the packet, calculate their own msm-ids which are larger than the one specified in the packet. The msm-ids thus increase as they radiate from the Sid. The nodes rebroadcast the NEW-SESSION message with the msm-id replaced by their own msm-ids. Each node is required to broadcast beacons to its neighbors. The beacon message contains the node id, msm-id, membership status, registered parent and child's ids and their msm-ids, and partition id. A node can join a multicast session by sending a JOIN-REQ. This JOIN-REQ is unicast to a potential parent node with a smaller msm-id than the node's msm-id. The node receiving the JOIN-REQ sends back a JOIN-ACK if it already is a member of the multicast session. Otherwise, it sends a JOIN-REQ.PASSIVE to its potential parent. If a node fails to receive a JOIN-ACK or receives a JOIN-NAK after sending a JOIN-REQ, it performs "Branch Reconstruction (BR)." The BR process is executed in an expanding ring search until the node succeeds in joining the multicast session. AMRIS detects link disconnection by a beaconing mechanism. If no beacons are heard for a predefined interval of time, the node considers the neighbor to have moved out of radio range. If the former neighbor is a parent, the node must rejoin the tree by sending a JOIN-REQ to a new potential parent. If the node fails to join the session or no qualified neighbors

exist, it performs the BR process. Data forwarding is done by the nodes in the tree. Only the packets from the registered parent or registered child are forwarded. Hence, if the tree link breaks, the packets are lost until the tree is reconfigured.

2.4 Core-Assisted Mesh Protocol (CAMP)

CAMP [5], [6], [7], [8] supports multicasting by creating a shared mesh structure. All nodes in the network maintain a set of tables with membership and routing information. Moreover, all member nodes maintain a set of caches that contain previously seen data packet information and unacknowledged membership requests. CAMP classifies nodes in the network as duplex or simplex members, or non-members. Duplex members are full members of the multicast mesh, while simplex members are used to create one-way connections between sender only nodes and the rest of the multicast mesh. "Cores" are used to limit the flow of JOIN REQUEST packets. CAMP consists of mesh creation and maintenance procedures. A node wishing to join a multicast mesh first consults a table to determine whether it has neighbors which are already members of the mesh. If so, the node announces its membership via a CAMP UPDATE. Otherwise, the node either propagates a JOIN REQUEST towards one of the multicast group "cores," or attempts to reach a member router by an expanding ring search of broadcast requests. Any duplex member of the node can respond with a JOIN ACK, which is propagated back to the source of the request. Periodically, a receiver node reviews its packet cache in order to determine whether it is receiving data packets from those neighbors, which are on the reverse shortest path to the source. If not, the node sends either a HEARTBEAT or a PUSH

JOIN message towards the source along the reverse shortest path. This process ensures that the mesh contains all such reverse shortest paths from all receivers to all senders. The nodes also periodically choose and refresh their selected "anchors" to the multicast mesh by broadcasting updates. These anchors are neighbor nodes, which are required to re-broadcast any non-duplicate data packets they receive. A node is allowed to discontinue anchoring neighbor nodes, which are not refreshing their connections. It can then leave the multicast mesh if it is not interested in the multicast session and is not required as anchor for any neighboring node. CAMP relies on an underlying unicast routing protocol. Routing protocols that are based on the Bellman-Ford algorithm cannot be used with CAMP, and CAMP needs to be extended in order to work with on-demand routing protocols.

Chapter 3

OVERVIEW OF THE PROTOCOL

3.1 Assumptions

Before talking about the protocol model, we first make some assumptions.

- Nodes know their location via GPS or by using a map
- Each node is identified with a unique ID number
- Each node gets information about neighbors from MAC layer support

3.2 Scenario

In the battlefield, some mobile soldiers cooperate with each other to try to destroy enemy force. Each soldier (may be a tank or aircraft, etc.) is represented as a node. And we assume that these nodes are equipped with radio to communicate with each other. And also there are some sensors deployed in the fields, which are responsible for detecting the threats and report their movement to soldiers. All the friendly nodes including the sensors form an ad hoc network.

We assume each threat has a round attack area, they can attack the friendly nodes that are within their attack radius. The friendly units and threats can be of different types such as aircraft, tank or infantry. The threats can also be clouds of poisonous gas. Friendly units can see the threats when they come

into their view, but that is not enough for them to make good preparation for a potential battle. It is highly helpful that friendly nodes can get information about the threats that are either directly threatening or are going to threaten them in a certain amount of time. The success of a battle largely depends on the efficient and accurate delivery of the warning information.

First we define time and space parameters for different types of units. We can imagine that it is necessary for a unit to know about a threat that can currently attack it, although the threat may be out of the sight of the unit. It'll be more useful if a unit can know about all the threats that are going to threaten it in a certain period of time, from its current location. For example, if a tank knows in advance the threats it will meet on its way in the following five minutes while it moves on, then it can make proper preparation for the potential conflicts. Here we assume all units of the same type use the same parameter. Each unit type can decide what time period to use, and that period of time is defined as the time parameter of the unit type. Likewise a space parameter defines the length of an area, and the threats in that area should be known by the unit in advance. Under such definitions, a faster moving unit tends to have more threats than a slower one. Thus they need to be informed with more warnings than the slower ones. Of the two parameters, we use the time parameter in our description and analysis. The space parameter is interchangeable with the time parameter. Under the definition of time and space parameters, a unit is said to be **directly threatened** by a threat if it is within the latter's attack radius. A unit is said to be **potentially threatened** by a threat if the threat is going to threaten the unit within its time or space parameter.

All the units can form an ad hoc network and exchange information. The source of the information is the sensor. Sensors are typically small devices that

have limited self-contained power supply. They are deployed in the battlefield to cover all or most of the battlefield. We assume different types of sensors for different types of threats. They detect threats within certain distance and can generate corresponding information in a format that can be understood by other units. They have transmission ability to reach other units to disseminate threat information.

3.3 Difference Between the Time-Space and Other Multicast Models

There is one distinction between our protocol model and those of the other ad hoc multicast protocols we talked about in the previous chapter. In a multicast protocol, one or more multicast groups are maintained. For each group, one or more nodes are the source or sender of multicast packets. The group members are the receivers of these packets. In those protocol models mentioned in the previous section, groups are well defined before hand, and a node knows exactly what group it wants to be in, and also whom they should ask in order to join the group. In other words, the receivers know about the identity of the sender. The sender, however, don't know what nodes are going to be in its group, thus they don't know about receivers' identity. In our protocol model, however, the receivers have no idea exactly which senders, here which are sensors, have information they need. And although the sensor can know roughly the region, in which the threatened units may exist, it is too costly for it to flood the information to all the units in that area. Therefore, in our protocol model, neither the sender nor the receiver knows the identity of each other.

3.4 Push/Pull - the Basic Strategy

The indeterminism in tactic information service makes it difficult to apply the existing ad hoc network multicast protocol. In the time-space multicast scheme, we use the push and pull to solve the indeterminism. The dissemination of the warning information is composed of the following steps: information collection, active information broadcast called push, active information requests called pull, and replies to pull requests. Under such a scheme, information can be delivered on-demand, and only to those nodes who need it. In the next part, we are going to describe these aspects of the protocol in detail.

Chapter 4

DETAILS OF THE PROTOCOL

4.1 Partitioning the Field

In order to make the message flow more easily controlled, we use a partition strategy in our protocol. The battlefield is divided into square blocks of equal size. Each unit is located in exactly one block. And each node can determine which block it belongs to, by its current location and the knowledge of the partitioning. In each block, one unit is elected to act as the "agent". The agent plays a crucial role in the dissemination of messages. We will see that when we go into more details. Figure 4.1 shows an example of the partitioning of a battlefield.

4.2 Election of Agent

Initially the agent can be elected in a simple way. A node with large transmission range can be a better candidate for an agent, but at the same time such a node tends to have a higher speed, which means that it crosses the border of blocks more frequently. Once the unit crosses the border it should transfer its agent duty to another unit in the old block. A slower agent may save some cost on block maintenance, but it tends to have less transmission ability. In our simulation, we simply select the node with the minimal ID as the agent.

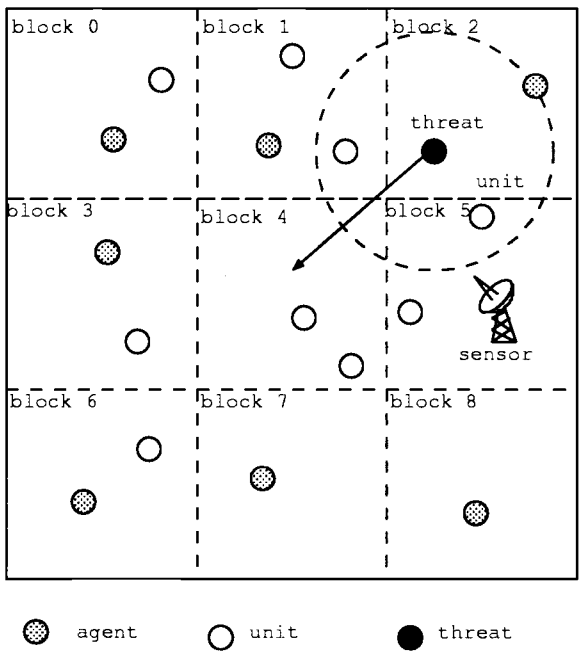

FIGURE 4.1: An example of partioning the field - 8 blocks

4.3 Agent Maintenance

When an agent moves out of the block to which it acts as the agent, it should no longer be the agent to that block. It tries to find another unit in the old block, and transfer the duty and necessary information to that node, and ask that node to take over the agent reponsibility. In our simulation, the new agent is selected from one of the old agent's neighbours in the previous block. We use this method in order to minimize the hops the critical information travels, so that the risk for the information to get lost during the transfer is reduced. The transfer needs only one hop from the old agent to the new one. One may think that since the old agent is near the border, the new agent selected in such a way would be near the border too, and have a higher chance of moving out

of the border itself. It is true that the new agent tends to be near the border, however because the moving direction of the new agent is random, we do not need to worry about that a new agent near the border has a higher chance to move out of the block sooner.

The agent acts as a bridge between the source and the receiver of a multicast group. Its responsibilities include:

- Collecting information about threats that will traverse its block, from sensors.
- Maintenance of a subscriber list
- Sending reply and updates to subscribers

The agent is selected among the units in one block in a simple way, for example, the unit with the minimal ID is selected to be the agent. Each unit in a block is informed of the location of the agent by periodic broadcast of a location update message from the agent. Thus each unit in the field knows about its agent's location.

4.4 Routing Algorithm

In the protocol we extensively use a geographic routing algorithm - MFR [15]. MFR stands for Most Forward with fixed Radius. The algorithm works like this: Suppose a source node S has a packet to send to destination node D. The source node S always tries to find a next hop so that the most advance toward the destination node is achieved by sending the packet to the next hop node, and so does every intermediary node who forwards the packet. In addition to

the distance to destination, we also take into account the transmission ability of the prospective next hop node, in selecting the next hop. We don't allow backward transmission in using MFR. It is obvious that MFR has one problem that is when a unit does not have any neighbor who is closer to the destination than the node itself, then the packet cannot be delivered.

Each node selects the next hop from its neighbors this way: If there are neighbors closer to the destination, the one has minimal value of the following expression is chosen to be the next hop: $d-r_t$, where d is the neighbor's distance to the destination node, and r_t is the next hop node's transmission radius. An example is shown in figure 2. S has three neighbors A, B and C, who have shorter distance to D than S itself. Although B is physically closer to D than A is, its transmission radius is shorter than that of A. So S ends up forwarding the message to A instead of B.

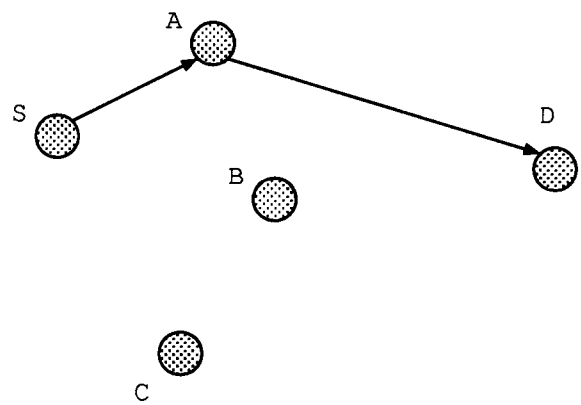

FIGURE 4.2: An example of MFR

4.5 Generating Threat Information

All the information about threats and their movement are generated by the sensors that are deployed in the battlefield. The sensors are so deployed that they cover the whole battlefield, while the intersection between the coverage of different sensors should be minimal. The sensors are the source of the multicast groups.

The sensors are resposible for detecting threats and creating warning messages in the first place. Different types of threats may require different sensors. We assume each type of threat node has one type of sensor for it. For certain types like the shooter threat type, it's impractical to have sensors for them, so we assign the sensor duty to friendly shooters themselves. A sensor may be stationary or mobile. It can send warning messages, and it can also listen to its neighboring units' broadcast but it typically does not accept any request messages, except that the shooters are normal units themselves so they can accept and forward messages. Each type of sensor has a search radius and a transmission radius. Typically a sensor has a larger search radius than a transmission radius.

4.6 Push Protocol

The warning messages are initially generated by the sensors when they detect threats. When a sensor or a collection of sensors detect the presence of a threat, they generate a limited broadcast message for nodes that are inside the *projected path of the threat.* Figure 4.3 shows an example. Here a tank threat is detected, the sensor(s) that detected the threat need to inform agent nodes that lie in the path of the tank of a possibly imminent attack by the tank. In the figure

we indicate that the extent of this warning push message extends out to blocks that lie within distance $s\tau_k$. Here s is the speed of the tank, τ_k is a constant and denotes the *time* specification that is used by tank sensors to determine the extent of the push.

Algorithm : *Push*

- When a sensor detects a threat it determines the projected velocity vector for the threat k. Let s denote the speed of the threat.

- Let τ_k represent the time specification used by the sensor to push information for the threat. The value τ_k might be different for different types of threats.

- The sensor (or sensors) send one THREAT_WARNING message to each of the agent node of the blocks that intersect within the push area with length $s(\tau_k)$ and width equal to the *strike range* of the threat. The push area is oriented in the direction of motion of the threat. The messages are sent with MFR.

- Each leader receiving the THREAT_WARNING message broadcasts it within their subscribe groups. The THREAT_WARNING message includes the nature of the threat, velocity, and any other relevant information (e.g., confidence level of the sensor in projecting the threat's motion).

- Whenever the threat moves a distance such that the furthest block warned is less than $s\tau_k$ distance away from the threat, a new push message is sent to blocks that are within $s(\tau_k)$ distance of the threat. In Figure 4.3, after

the tank moves through one (or two) blocks in the indicated direction, a new push is generated to cover blocks a, b, c and d.

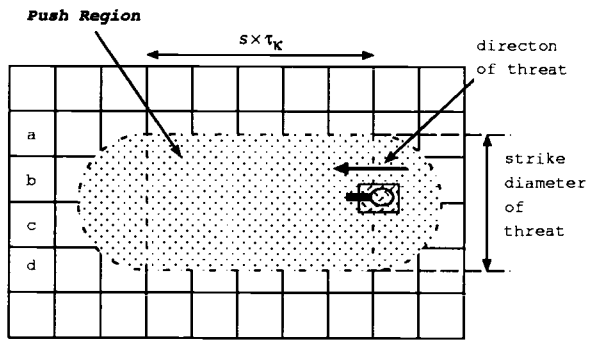

FIGURE 4.3: An example of push

4.7 Pull Protocol

Pull is the second step in disseminating the warning messages. As we have mentioned before, the sender does not know about the receivers' identity, that is, the sensors do not have any information about which units need the messages it generates. They just push the warnings forward for a certain range, according to the movement of the threat. The necessity to the message is based on the threat and the unit's location and movement. And also, neither do the agents know about which units need the messages they have got from the sensors. So push itself is not enough to deliver the message to units who need them. A simple yet very costly way is to use flooding - for every warning message, flood it to every unit in the battlefield. Obviously we can do better than that.

The sensor knows about the nature and movement of the threats. After getting push messages from the sensor, the agents accumulate the knowledge about the threats too. Since only the units themselves know about their own necessity and their mobility, it is a good idea for the units themselves to initiate the second step of the information dissemination. Thus we have the pull. Under the pull strategy, the units send out pulls to agents ahead of them, which includes what types of threats it is interested in, its own location, its direction and speed. Using this information, the agents know how to spread the warning messages they get from sensor further to the subscribing units.

Pull Area It is crucial to find the set of agents that may have threat messages useful to a node. The node sends a pull to those agents and the agent will reply with those messages. Let t_n be the time parameter of a node (please see previous sections for definition), and τ be the time used by a sensor to send push messages. It is obvious that one pull should be sent to each of the agents whose block the node will pass during t_n which the time specification of the node. However, this is not enough because after t_n, threats may reach the same block as the node does, while the agent in that block does not necessarily have a warning message for that threat. Thus we should expand the push area. What we do here is to include into the push area a circlular region which is centered at the point which the node is going to be after t_n, and has radius $s^{max} * (t_n - \tau)$, where s^{max} is the maximum threat speed. This guarantees that the node knows all the threats it will meet after t_n. Similarly at all time before t_n, the pull area should be expanded in the same way, except for the first τ, during which all threats are already known by the agents whose the node will pass. For simplicity, here we further expand the pull area into a fan area which

is centered at the node's current position, and includes the expanded circle at time t_n. Thus the area to which pulls should be sent is actually a fan, with radius r and angle θ.

Let s_n be the speed of the node. Then the radius of the fan area is

$$r = s_n * t_n$$

Let s_k^{max} be the maximum of threat's speed. Then the radius to expand the region at time t_n is:

$$d = s_k^{max} * (t_n - \tau)$$

Where s_k^{max} is the speed of the fastest threat, t_n is the time parameter of the node, and τ is the time specification of sensor.

So the angle of the fan is:

$$\theta = asin(d/r)$$

Figure 4.4 illustrates the calculation of the pull area.

The fan area we calculated ensures that a node gets all threats that it will meet for t_n. As the node moves on, there will be new area that is not covered by the fan area. So the new pull messages should be sent to agents in those new blocks. However, because the agents which have already received the node's pull message in last pull do not need to receive a new pull because they still keep the old one. The area that needs the pull is the difference of two consecutive fans. For simplicity, we use an arc-shaped area, which is illustrated in figure 4.5.

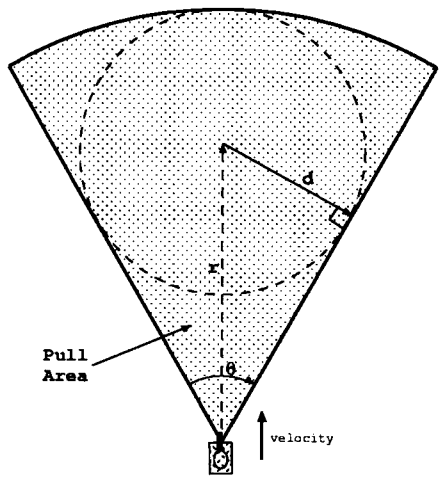

FIGURE 4.4: pull area calculation

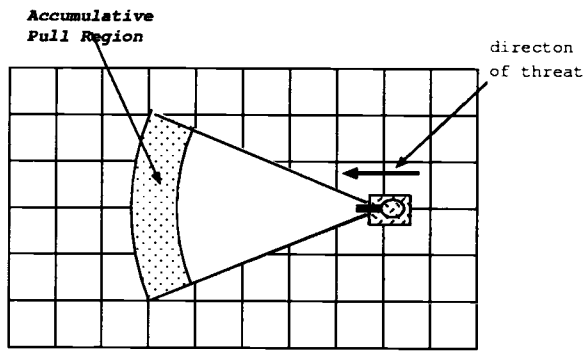

FIGURE 4.5: Accumulative Pull

Algorithm : *Pull*

- Let a node's *time* specification be t_n (the same algorithm works if we use a node's distance specification). In other words, the node needs to

be warned of all threats it considers threatening that it could encounter within time t_n.

- Let the node's speed be s_n. Therefore in time t_n it will be located in a block that is $s_n t_n$ distance away.

- The node start pulling with sending a pull to a fan area, which is centered at its current position, with $s_n * t_n$ as the radius, and $2 * asin\frac{s_n * t_n}{s_k^{max} * (t_n - \tau)}$ as the angle.

- after each fan-shaped pull, when the node moves the distance equal to the size of a block, it sends a pull to the arc-shaped area, with radius $s_n * t_n$ and $s_n * t_n - b$, where b is the size of the block.

- When a node changes its direction, it sends a new fan-shaped pull.

4.8 Sticky Pull

Every agent records the pull messages it gets. After an agent gets a pull message from a unit, it stores the message into its pull message list. The sender of the pull becomes one of the subscribers of the information the agent keeps. If there's already a pull message from the same unit, it keeps the one with larger sequence number and discards the other one. In this way, only one most up-to-date entry is kept for one single unit. Each entry is kept for a certain amount of time in the list, then it is deleted. The time is computed based on the unit's speed, direction and distance from the agent. Its value should be such for the unit to reach and pass the block which the agent is in. If the unit changes its direction but the block is still within its pull area, a new pull will be sent to the agent

and the agent will update the entry for that unit. However if the unit's new pull area does not contain the block, the agent is going to keep a useless entry for a certain amount of time, during which the agent will send reply messages to the unit, which is usually not useful for it any more. Fortunately this entry will be deleted after it expires.

The pull message contains the location, speed and direction of the subscriber. One of the agent's duties is to renew the information so that it always know each subscriber's correct location, which is crucial for sending reply or update information to the subscriber. Basically the agent can periodically scan the pull list and update all the entries in the list. Another way is to tag a timestamp to each entry marking when that entry was last updated. When that entry is going to be used to send updates, the location is updated according the time lapsed since the stamp made, and the stamp is also renewed.

Each agent acts as a multicast group manager, and each unit who sends pull message to it subscribes to the information the agent gets. Here we can view the subscribers to one agent to be a multicast group. Multiple units can join in one group, while one agent maintains only one group. An agent can itself be a subscriber to one or more multicast group(s). The agent is responsible for keeping its subscriber updated with new warning messages.

4.9 Reply/Updates - Completing the Dissemination

To complete the dissemination of the information, the agents forward the warning messages to its subscribers. The agents send replies either on receiving pull requests, or they send update messages on a periodic basis.

In order to minimize the size of reply or update message, for each pull

request, the agents use a filter to select those messages the sender of the pull request is interested in. A simple geometric algorithm is used to pick out those messages to include. And also the messages are packed into one single reply or update message to send back to the subscriber. For the periodic update, the frequency depends on the speed of the threat. A timer is used by each agent to trigger update sending.

4.10 Ally Information Dissemination

Besides the information about threats, it is also useful to have the nodes know about the location and movement about other friendly nodes. With the basic protocol infrastructure for threat information, we can add this part quite easily. It makes sense that a unit has the same requirement for friendly nodes like that for threats, with the same time and space parameter. So the push/pull scheme can be used for spreading friendly node information as well.

4.10.1 Pushes

In threat information dissemination, the source of the information is the sensors. While in the friendly node information dissemination, the source of the information is the friendly nodes themselves. Like the sensors, all friendly nodes send push messages to agents whose blocks intersect with the push area. The push messages contain in them the location, speed and direction of the units. The frequency of such pushes can be set to such value that one push is sent during the time that takes a unit to traverse a block. So the frequency is calculated based on the unit's speed and the block size. The agents keep the list of such pushes and update the location information periodically or on-demand.

4.10.2 Pull, Reply and Update

Since we assume the same pattern of pull is used to get friendly node information, with the same parameters, there is no need for another separate pull message for friendly node information request. Once the agents get the push messages from the friendly nodes, they can reply the subscribers with friendly node information. The messages for friendly node information can be merged in the reply or update message for threat information. And similarly separate frequencies can be calculated for the friendly node information.

Chapter 5

PERFORMANCE EVALUATION

We evaluated performance of our T-S multicast protocol via extensive simulations. A simulation program is implemented in Java. We vary the simulation parameters in different experiments in order to find how the result responds. Each experiment is repeated for 10 times, to get the mean value and variation of the results. In the following we first take a brief look at the simulation program, and then we will present and analyze the experimental results.

5.1 The Simulation Program

We used general-purpose language in building the simulation program. A simulation program built with general-purpose language has the advantage of more efficiency, flexibility and portability. In order to get maximum flexibility and expandability in the simulation, we used OOD/OOP in our simulation implementation. The advantage of OO enables us to add new protocol components and change the behavior of simulation objects easily. Further for maximum portability, we select Java because it is available across many platforms.

Classes are defined for each component of scenario. Here we give a brief description for classes.

There are 5 classes for nodes. The hierarchy is shown in figure 5.1.

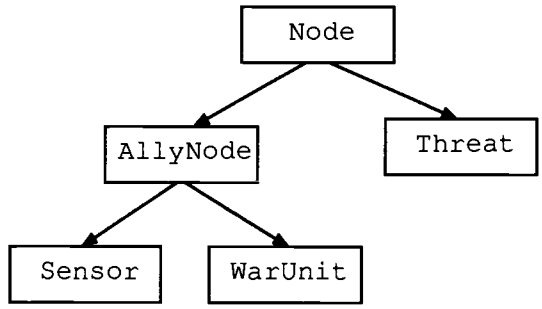

FIGURE 5.1: Node Classes Hierarchy

■ **Class Node** This class defines the base class for all nodes in the simulation. Basic parameters like position, speed, and direction are defined as member variables, and access methods are defined for these variables.

■ **Class AllyNode** AllyNode inherits class Node, and is the parent class of Sensor and WarUnit class. In addition to those inherited from Node, it defines some features shared by allied nodes, like a neighbor list vector to hold current neighbor nodes, a sequence number counter, transmission radius and speed. Two examples of methods defined in this class include $mpForward()$, which implements the MFR, and $push()$, which generates and sends out warning messages.

■ **Class Sensor** Class Sensor inherits Class Node, and represents the sensor nodes. Parameters for sensors like detect radius and time parameters for calculating push area are defined. Note that the search() and push() methods are all defined in AllyNode class instead of the Sensor class, because we need shooter nodes to be able to detect shooter threats by themselves.

■ **Class WarUnit** This is the class for all other allied nodes. More specific member variables are defined, such as message vectors used to hold threat or friendly node information, time/space parameters of the node, node type(tank/shooter) and some flags. A lot of message handling methods are defined here. Each node should help forward messages, as well as sending push/pull and receiving reply messages. Care should be taken in order to avoid errors like "racing" in message dissemination.

■ **Class Threat** This is the class for threats. It is the other child class of AllyNode class.

■ **Class Message** The Message class is meant for all push/pull/reply messages. The type field identifies which category it belongs to. Here we do not define separate class for each type of message, in order to improve the simulation speed. The reason is that before processing a message, a node should know what type the message is of. It is costly to check what class an object is of, so we merged all message types into one single message class and use the type field to tell them apart.

■ **Class TransTime** One thing worth mentioning here is about memory space. During the simulation run, a great number of messages are created, replicated and forwarded. The total memory usage explodes shortly after simulation begins. Thus we apply a pointer-like strategy in handling copies of messages. Most messages in the system share some components. For example when a sensor detects a threat, it creates a message and sends one copy to each agent in the push area. The copies are mostly the same except the destination address. It would save a lot of space if we extract the different part into a small

object, and share the common part in just one Message object. That is how we use the TransTime class - a helper class associated with the Message class in order to save memory space.

■ **Class SimDaemon** The SimDaemon class is in charge of starting and controlling the simulation run. It creates all the nodes in the simulation and set them off. It also coordinate with the display classes - SimGui and Display to have the battlefield scenario displayed in real time.

Also some helper classes are defined. The *Calc* class takes care of all mathematical calculations needed in disseminating the messages, by providing static methods. The *SimSetting* class stores all simulation parameters, and provides the support for simulation setting files.

5.2 Performance Metrics

In order to quantify the performance of our protocol, we concentrated on two metrics:

1. *Message overhead:* The metric here is the number of messages/second/node that are exchanged in the course of the simulation run. This includes all push/pull/reply messages as well as control messages exchanged in order to maintain the blocks.

2. *Coverage:* It is possible that some nodes do not receive threat warnings, due to routing failure, error in calculation or other reasons. Thus, we measured the percentage of nodes that ought to have been warned of a threat but were not. Ideally we would like this number to be zero.

However, periodic network partition and routing failure in MFR make it difficult to achieve this value.

The remainder of this section is organized as follows. We first describe the experimental set up and then describe our experimental results.

5.3 Simulation Parameters

For the simulation we assume that the battlefield is a 10km square region. The threats consist of tank, shooter and gas threats. The allied forces have tanks and shooters. Tanks move at an average speed of 72kmph, shooters move at an average speed of 9.6kmph and wind speed is constant at 18kmph. The battlefield has sensors capable of detecting tank and gas threats. In the simulations we use 64 gas sensors and 64 tank sensors evenly distributed throughout the battlefield. The sensors cover the whole field. A gas sensor can detect a gas threat within 1km and a tank sensor can detect tank threats within 1km as well. Shooter threats can only be detected by other (friendly) shooters. Thus, it is possible that a shooter threat may go undetected because of human error. We use a probability of 0.2 that a shooter threat will not be detected. Finally, we assume that tanks, shooters and sensors have radio capability. The transmission radius of the sensors is assumed to be 2km, tanks have a transmission radius of 5km and a shooter can transmit to a distance of 1km. The data rate available is 1Mbps.

The time-space parameters we use were the following. Tanks needed to know about tank threats within 6km and shooters needed to know about tank and shooter threats within 900m. Each simulation is run for 300 seconds of real-time (a couple hours of simulation time) and the simulation time step is 5 milisec

of real-time. We run each case ten times and compute 95% confidence values. In each case the confidence intervals were very tight (less than 5% of the point values).

5.4 Discussion of Results

In all the following plots we use a constant number of enemy threats. Specifically, we use 10 tank, 10 gas and 30 shooter threats. In Figure 5.2 we plot the total message overhead as a function of the number of friendly tanks (no shooters) and in Figure 5.3 we plot the message overhead as a function of the number of shooters (no tanks). In Figure 5.2 we see that the message overhead increases slightly as the number of nodes increases. And in figure 5.3 we see that the overhead increasing is sharper when the number of shooters increases from 100 to 300, while after 300, the increasing slow down. The explanation for the sharp increase in smaller node number section is that when the number of shooters is small (less than 300 in the 10km field), the connectivity between nodes is so poor that many messages cannot be delivered by the MFR algorithm we used in simulation. The lower overhead does not mean that the protocol is more efficient for small number of nodes. The lower success rate in these small number configurations is a proof, which we will see below. An interesting observation is that the message overhead increases sub-linearly with increasing numbers of friendly nodes. This indicates that our T-S protocol is scalable to large networks. Intuitively this makes sense since the extent of a sensor's multicast is geographically limited to nodes in immediate danger. Thus, even if the network size grows, the message overhead ought to remain the same.

Figure 5.4 plots the success rate on threat warning as a function of the

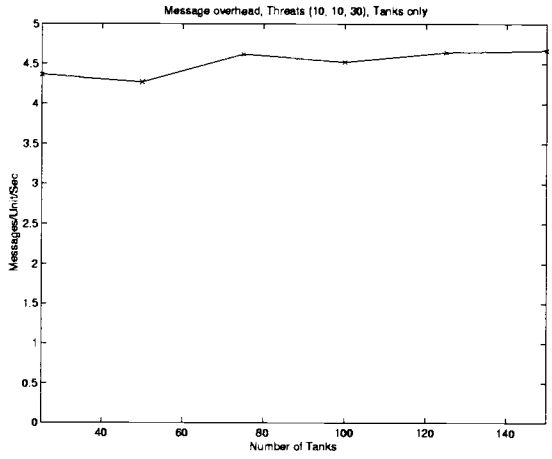

FIGURE 5.2: Message overhead – Tanks only.

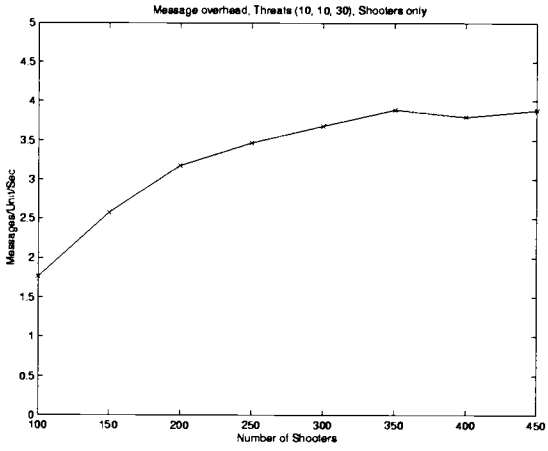

FIGURE 5.3: Message overhead – Shooters only.

number of tanks. The success rate for the tank only case remains above 90% for the six number settings. However for the shooter only case, only about 67% of the shooters get threat information in time for 100 shooter. The percentage increases quickly to more than 95% when the number of shooters increases to

250. The explanation is that pull messages do not always reach the desired blocks because the network does not have a geographically direct path (we use MFR, a geographical routing algorithm to send messages so if there is no next hop in the direction of the destination, the packet is dropped). Furthermore, the push and reply message delivering also relies on the same MFR, so they tend to have a higher failure rate at smaller number configuration. However, as the number of shooters increases, the probability of finding a path increases and hence the percentage of nodes warned in time also increases. In the case of tanks, their greater transmission radius ensures that there is a greater probability of finding routes.

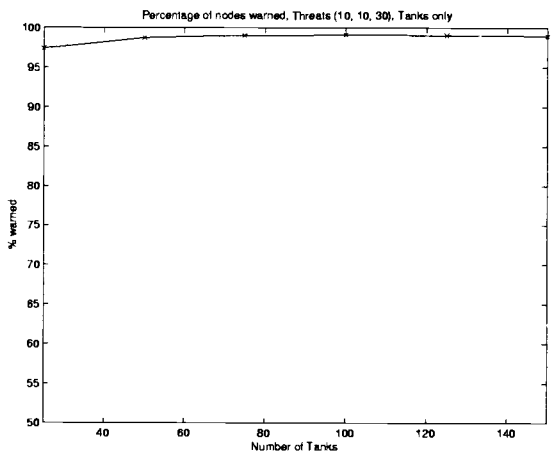

FIGURE 5.4: Threat Information Success Rate – Tanks only.

Apart from the warning of threats, we also experimented on the friendly information dissemination. Figure 5.6 plots the success rate on friendly information checking, as the function of the number of tanks, and figure 5.7 plots

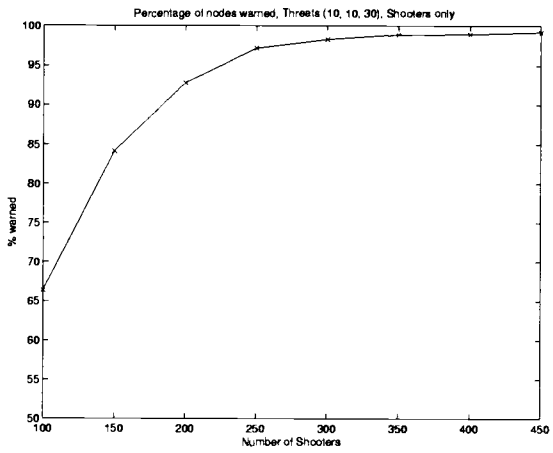

FIGURE 5.5: Threat Information Success Rate – Shooters only.

the success rate as a function of the number of shooters. For the tank only case, we can see that the success rate is similar to that of threat information, while for shooter only case, the success rate is generally lower than that for threat information. The difference is caused by the following facts: the same algorithm is used for disseminating both threat and friendly node information. The difference lies in the source of message. For threat information, one shooter may be detected by more than one friendly shooter, thus more than one message may be sent. For friendly node, each node itself sends message about itself to agents, so if there is a routing failure, no other friendly node can help it. In short, the friendly node information success rate is more affected by routing failure. That is why the friendly information dissemination has a lower success rate in the shooter case. For the tank only case, because of the large transmission radius of tank, the routing failure rate itself is much lower than the shooter only case, so the friendly node information success rate in this case is not negatively influenced so obviously. Running the simulation in the debugging mode also reveals

that most of the notification failure of friendly node is due to routing failure.

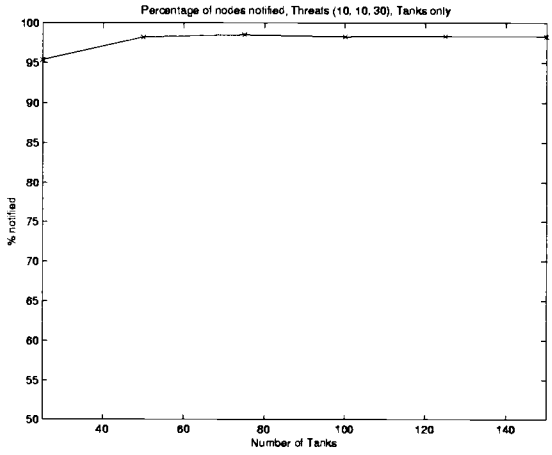

FIGURE 5.6: Ally Force Information Success Rate – Tanks only.

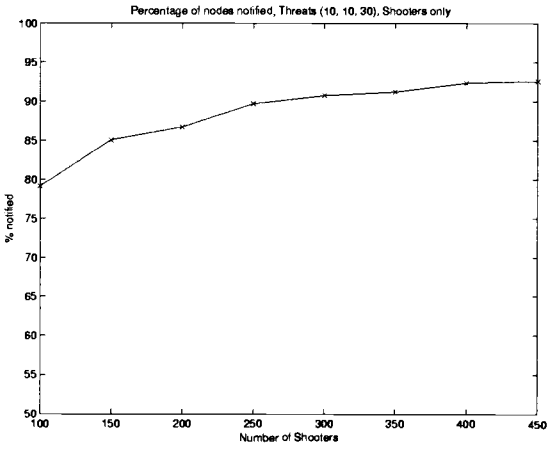

FIGURE 5.7: Ally Force Information Success Rate – Shooters only.

To find how the block size influence the result, we run some experiments with different block size settings. Figure 5.8 plots the threat information success rate as a function of the block size. Figure 5.9 plots the message overhead as a function of the block size. Figure 5.10 plots the friendly node information notification success rate as a function of the block size.

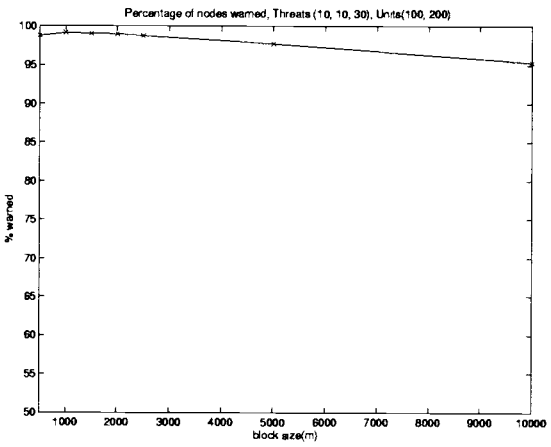

FIGURE 5.8: Threat Information Success Rate for Different Block Size.

First, it is noteworthy that smaller block sizes result in greater message overhead. This is because the same push area covers more blocks and so more push messages are sent. Similarly, the same pull area covers more blocks so more pull messages are sent. Further, because each node subscribes to more agents, more agents are sending reply or update messages back to that one node. Finally, the overhead of maintaining blocks is also greater (since the time a node spends in a smaller block is smaller). Second, we can see that the block size does not affect the success rate much. Figure 5.8 shows that the threat information success rate peaks at around 1000 meter block size and

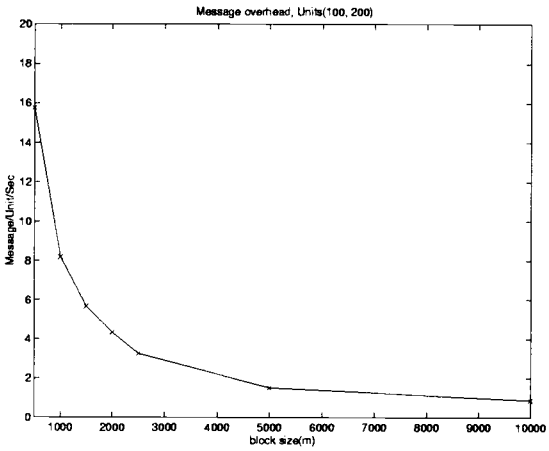

FIGURE 5.9: Message Overhead for Different Block Size.

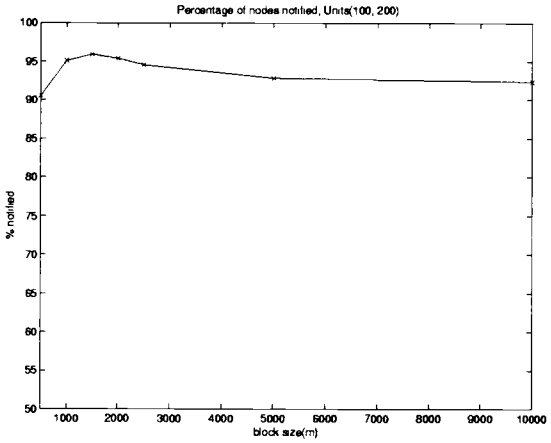

FIGURE 5.10: Ally Force Information Success Rate for Different Block Size.

drops slightly when the block size increase. For the extreme case of 10000 meter block size, where the whole battlefield is one single block, the success rate is a little above 95%. The reason for smaller block to have higher success rate is that more agents receive threat information packets, a unit subscribes to more

agents, thus a unit will have higher chance to get a warning in the end, for a particular threat. Yet the success rate does not vary much.

In figure 5.10 we see similar trend in the curve for ally force information success rate. The same explanation works for this case. Meanwhile we can see that the success rate for ally force information is lower than that for threat information. Again the explanation for the lower ally force information rate for shooter only case in figure 5.7 can be used here.

Figure 5.9 shows great difference in message overhead from different block size. Here it takes on average about 15 message for each node in one second, while the number reduce to less than 4 for 2500 meter block size. Further when the whole battlefield is one single block, the overhead shrinks to less than 2. So message overhead favors larger block size. On the other hand, larger block size means fewer agents and more centralized management of information, which also brings higher risk in a real world battle site. Consider that an agent is captured and maneuvered by the enemy. So there will be a trade off between message overhead and security consideration.

BIBLIOGRAPHY

[1] Y.C. Tay, C.W. Wu and C.-K. Toh. *Ad Hoc Multicast Routing protocol utilizing Increasing id-numberS (AMRIS) Functional Specification.* Internet-Draft, draft-ietf-manet-amris-spec-00.txt, Nov. 1998, Work in progress.

[2] S.E. Deering and D.R. Cheriton. *Multicast Routing in Datagram Internetworks and Extended LANs.* ACM Transactions on Computer Systems, vol. 8, no. 2, May 1990, pp.85-110.

[3] A. McAuley, E. Bommaiah, M. Liu and R. Talpade. *AMRoute: Ad Hoc Multicast Routing Protocol.* Internet-Draft, draft-talpade-manet-amroute-00.txt, Aug. 1998, Work in progress.

[4] E.D. Kaplan (Editor). *Understanding the GPS: Principles and Applications.* Artech House, Boston, MA, Feb. 1996.

[5] J.J. Garcia-Luna-Aceves and E.L. Madruga. *The Core-Assisted Mesh Protocol.* IEEE Journal on Selected Areas in Communications, vol. 17, no. 8, Aug. 1999, pp. 1380-1394.

[6] J.J. Garcia-Luna-Aceves and E.L. Madruga. *A Multicast Routing Protocol for Ad Hoc Networks.* In Proceedings of IEEE INFOCOM'99, New York, NY, Mar. 1999, pp. 784-792.

[7] E.L. Madruga and J.J. Garcia-Luna-Aceves. *Multicasting Along Meshes in Ad-Hoc Networks, In Proceedings of IEEE ICC'99.* Vancouver, Canada, Jun. 1999, pp. 314-318.

[8] E.L. Madruga and J.J. Garcia-Luna-Aceves. *Scalable Multicasting: The Core Assisted Mesh Protocol.* To appear in ACM/Baltzer Mobile Networks and Applications, Special Issue on Management of Mobility, 1999.

[9] J. Moy. *Multicast Routing Extensions for OSPF.* Communications of the ACM, vol. 37, no. 8, Aug. 1994, pp. 61-66, 114.

[10] D. Farinacci, V. Jacobson, C.-G. Liu, S. Deering, D.L. Estrin and L. Wei. *The PIM Architecture for Wide-Area Multicast Routing.* IEEE/ACM Transactions on Networking, vol. 4, no. 2, Apr. 1996, pp. 153-162.

[11] M. Gerla, S.-J. Lee and C.-C. Chiang. *On-Demand Multicast Routing Protocol.* In Proceedings of IEEE WCNC'99, New Orleans, LA, Sep. 1999, pp. 1298-1304.

[12] W. Su, S.–J. Lee and M. Gerla. *Ad hoc Wireless Multicast with Mobility Prediction.* In Proceedings of IEEE ICCCN'99, Boston, MA, Oct. 1999, pp. 4–9.

[13] W. Su, S.–J. Lee and M. Gerla. *On–Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks.* Internet–Draft, draft–ietf–manet–odmrp–01.txt, Jun. 1999, Work in progress.

[14] P. Francis, T. Ballardie and J. Crowcroft. *Core Based Trees (CBT) – An Architecture for Scalable Inter–Domain Multicast Routing.* In Proceedings of ACM SIGCOMM'93, San Francisco, CA, Oct. 1993, pp. 85–95.

[15] Victor O.K. Li and Ting–Chao Hou. *Transmission Range Control in Multihop Packet Radio Networks.* IEEE Transactions on Communications, vol. COM–34, No. 1, Jan. 1986, pp. 38–44.